CS/ENGRD 211
Fall 2005

Lecture 1: Overview

http://www.cs.cornell.edu/courses/cs211/2005fa

1

Course Staff

- Instructors:
  - Professor Paul Chew
  - Professor David Schwartz
  - Lead lectures and coordinate course
- Administrative Assistant:
  - Kelly Patwell
  - General administration
- Locations, office hours, contact info?
  - See Staff on website

2

Student Course Staff

- Teaching Assistants:
  - TAs lead recitation sections
  - TAs are your main contact point
- Consultants:
  - In Upson 360, hours TBA online
  - "Front line" for answering questions
- More info?
  - See Staff on website

3

Lectures

- TR 10:10-11am, Olin 155
- Attendance is mandatory
- Lecture notes will be online—print them before class and bring them to class
- We will occasionally make small last minute changes to the notes, so don't print them too far in advance
- Readings will be posted online together with lecture notes

4

Sections

ENGRD Course ID	COM S Course ID	Section	Day	Time	Place	Instructor(s)
734-194	675-107	1	T	12:20-1:10	HO 110	TBA
734-285	675-194	2	T	1:25-2:15	HO 206	TBA
734-382	675-489	3	T	2:30-3:20	HO 110	TBA
734-449	675-490	4	W	12:20-1:10	HO 219	TBA
734-551	675-549	5	W	1:25-2:15	BD 140	TBA
734-580	675-574	6	W	2:30-3:20	UP B17	TBA
734-643	675-746	7	T	12:20-1:10	HO 401	TBA
734-734	675-884	8	W	1:25-2:15	OH 165	TBA

- Attendance is mandatory
- Usually review, help on homework
- Sometimes new material

5

CS212

- ***CS 212: Java Practicum***
- 1 credit project course
- Substantial project
- 1 lecture per week
- Required for CS majors; recommended for others
- Take 211 and 212 in same semester?

6

Obtain Java

- We do not require an IDE
- We generally use Dr Java
- We do require Java 5
- See **Help & Software** under **Java Resources** on website

7

Java Help

- CS 211 assumes basic Java knowledge:
  - control structures
  - arrays, strings
  - classes (fields, methods, constructors)
- Need review?
  - Tutorials, links on website (**Help & Software**)
  - ***Java Bootcamp***:
    - self-guided tutorial—material on website
    - You can also work with staff on it: 7:30-10:30pm on both Tue 8/30 and Thu 9/1 in Upson B7
    - Same material on both days

8

Academic Excellence Workshops

- Two-hour labs in which students work together in cooperative setting
- One credit S/U course based on attendance
- ENGRG 210, 745-791, Fridays, 2:30-4:25, ACCEL
- See CS211 web site for more info

9

Course Work

- 6 assignments involving both programming and written answers
- A.I. check each homework assignment
- Two prelims and final exam
- Course evaluation

Assignments (44%)						Exams (55%)			Eval (1%)
A1	A2	A3	A4	A5	A6	P1	P2	F	E
6	7	7	7	7	10	15	15	25	1

10

Assignments

- Assignments may be done by teams of two students (except A1)
- You can do them by yourself if you like
- Finding a partner: choose your own or contact your TA
- Monogamy is strongly encouraged, polygamy is strongly discouraged, and divorces are permitted in case of irreconcilable differences
- See website course info and Code of Academic Integrity

11

CS211 Objectives

- Concepts in modern programming languages:
  - recursion, induction
  - classes, objects
  - inheritance, interfaces
- Efficiency of programs
- Data structures: arrays, lists, stacks, queues, trees, hash tables, graphs
- Software engineering: How to organize large programs

This is not a course on Java programming!

12

Lecture Sequence

- Introduction and Review
- Induction and Recursion
- Grammars and Parsing
- OOP Revisited
- Lists and Trees
- Inheritance and Interfaces
- Algorithm Analysis
- Asymptotic Complexity
- Searching and Sorting

13

More Lecture Topics

- Generic Programming
- Sequence Structures: stacks, queues, heaps, priority queues
- Search Structures: binary search trees, hashing
- Graphs

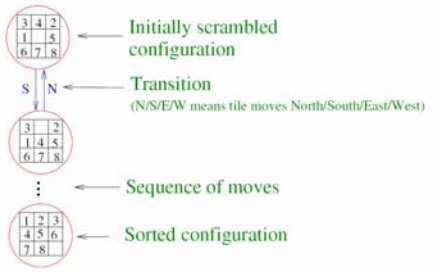
14

Some Examples

- CS211 lectures typically use these two examples:
  - 8-Puzzle Game
  - SaM
- See links in Lecture Notes on website

15

Sam Loyd's 8 Puzzle

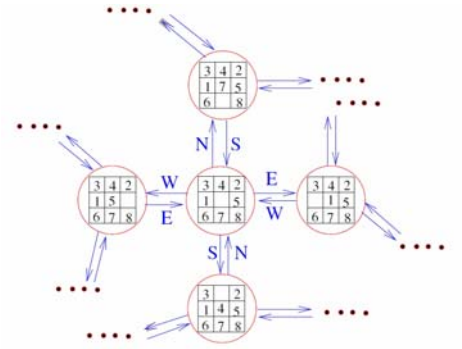


Goal: Given an initial configuration of tiles, find a sequence of moves that will lead to the sorted configuration.

A particular configuration is called a ***state*** of the puzzle.

16

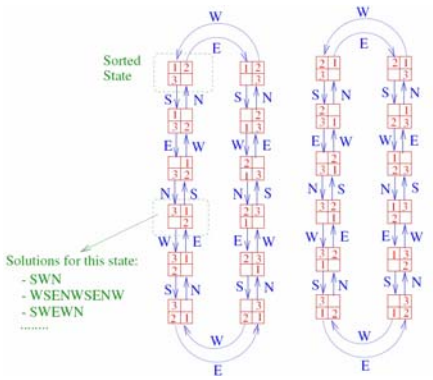
State Transition Diagram of 8-Puzzle



State Transition Diagram: picture of adjacent states.
A state Y is ***adjacent*** to state X if Y can be reached from X in one move.

17

State Transition Diagram for a 2x2 Puzzle



18

Graphs

- State Transition Diagram in previous slide is an example of a ***graph***
- Graph has ***vertices*** (or ***nodes***): in our example, these are the puzzle states
  - ***edges*** (or ***arcs***): connections between pairs of vertices
  - vertices and edges may be annotated with some information
- Other examples of graphs: airline routes, roadmaps, . . .

19

Path Problems in Graphs

- Is there a path from node A to node B?
- What is the shortest path from A to B?
- Traveling salesman problem
- Hamiltonian cycles
- . . . will see later in semester

20

Simulating 8-puzzle

- What operations should puzzle objects support?
- How do we represent configurations?
- How do we specify an initial configuration?
- What algorithm do we use to solve a given initial configuration?
- What kind of GUI makes sense for puzzles?

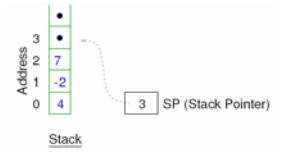
21

SaM

- ***SaM*** is a simple StAck Machine:
  - Modeled roughly after the Java Virtual Machine (JVM)
  - Use it to understand how computers work at the assembly language level (.class file level)
  - Use it to understand how compilers work
- Download it from course homepage
- Used extensively in CS212

22

SaM's Stack



Note: For now, assume only integers can be pushed on stack. SaM actually allows floats, characters, etc. to be pushed, and it tracks type of data. GUI displays type (I:integer,F:float,. . .), but ignore this for now.

Stack: an array of integers
- Stack grows when integer is "pushed" on top.
- Stack shrinks when integer is "popped" from top.
- Stack starts at address 0 and grows to larger addresses.

Stack pointer (SP):
- first "free" address in stack
- stores integer address
- initialized to 0

23

Some SaM Commands

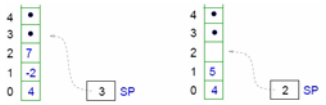
- All arithmetic/logical operations pop values from stack, perform operation, push result, and move SP to first free address
- Some commands:

```
PUSHIMM int
      // push integer int onto top of stack
ADD
      // pops two values from top of stack
      // adds them and pushes result
SUB
      // pops two values (say top and below)
      // and pushes result of doing (below - top)
TIMES
      // works like ADD
GREATER
      // Boolean values are simulated using 0/1 (false/true)
AND
      // logical AND
STOP
      // terminate execution of program
```

24

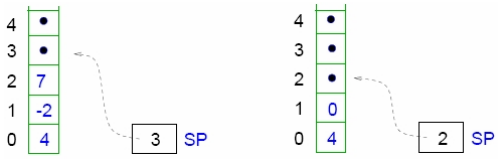
Demonstrate SaM Commands

PUSHIMM 16



ADD:
- Pop 7
- Pop -2
- Determine 7 + (-2)
- Push result

Booleans and SaM



Booleans are simulated in SaM with integers:
- False → 0
- True → any int except 0 (usually 1)

GREATER:
- Pop two values (V_{top} and V_{below}) from stack (V).
- So, V_{top} = 7 and V_{below} = -2.
- If $V_{below} > V_{top}$ push 1; else push 0.
- In example, we would push 0.

SaM Programs

- Example 1:

```
PUSHIMM 5
PUSHIMM 4
PUSHIMM 3
PUSHIMM 2
TIMES
TIMES
TIMES
STOP // should leave 120 on top of stack
```

- Example 2:

```
PUSHIMM 5
PUSHIMM 4
GREATER
STOP //should leave 1 on top of stack
```


SaM Simulator

- What operations must SaM objects support?
- How do we represent the internal state of SaM?
- How do we load programs from a file?
- How do we write code to interpret each of the opcodes?
- See “Chapter 1” in CS212 lecture notes

Why bother?

- By the end of CS211, you will be able to design and write moderately large, well-structured programs to simulate such systems.
- Why do we care about large, well-structured programs written in modern languages such as Java?
- Why are data structures such as trees and graphs and algorithms (e.g., quicksort) and theoretical issues (e.g., computational complexity) so important?

Moore’s Law

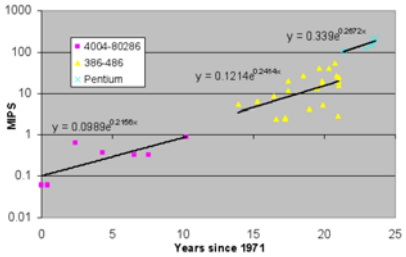


Figure 5: Processor performance in millions of instructions per second (MIPS) for Intel processors, 1971-1995.

From *Lives and death of Moore’s Law*, Ilkka Tuomi, 2002

Grandmother's Law

- Brain takes about 0.1 second to recognize your grandmother
  - About 1 second to add two integers (e.g. 3+4=7)
  - About 10 seconds to think/write statement of code
- Your brain is not getting any faster!

31

Motivation

- Computers double in speed every 18 months
  - Software doubles in size every M Years
  - Data doubles in size every N Years
  - Your brain never doubles in speed
  - But we do get smarter, and can work in teams
- Computer science gets better:
  - Better algorithms
  - Better data structures
  - Better programming languages
  - Better understanding of what is (and is not) possible

32